TECHNION, ISRAEL INSTITUTE OF TECHNOLOGY
Department of Mechanical Engineering

Robot Navigation

Part 1: Mini-Project

Avital Dell'Ariccia 941170276
Leeor Ravina 316399542
Benjamin Barnes 941180010

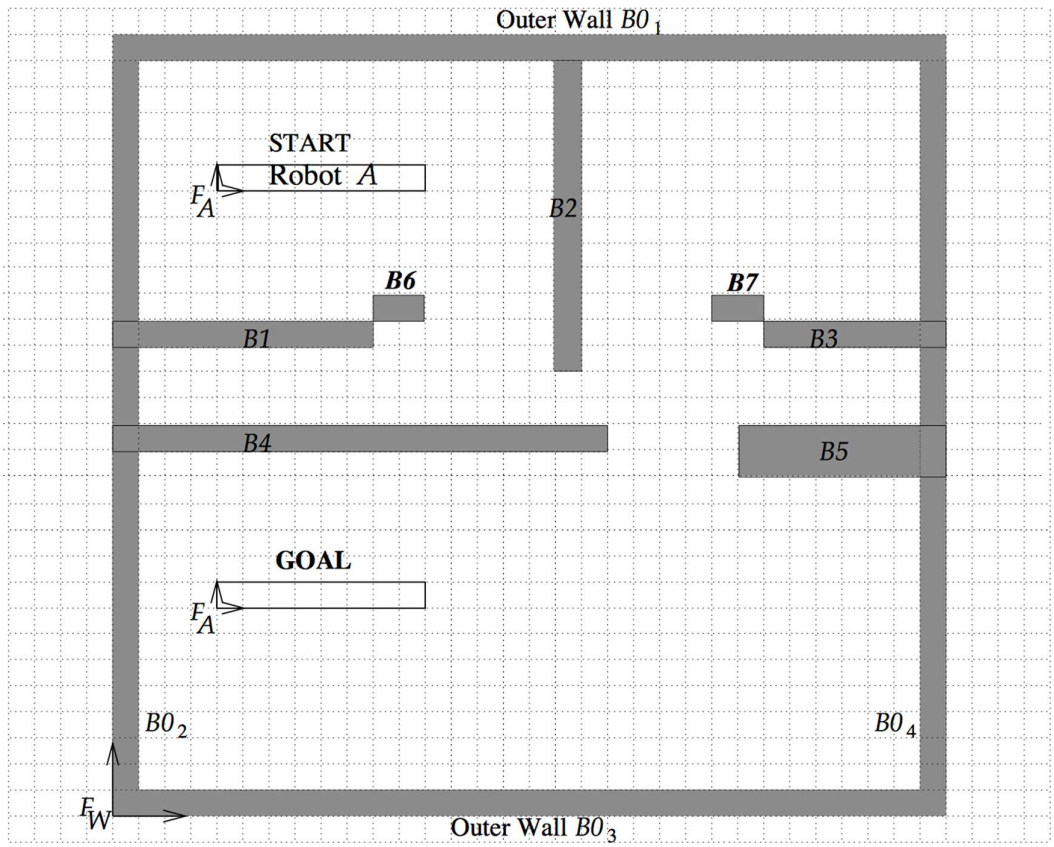


Figure 1: The Bed Movers Problem.

Part 1

Write a program that implements the algorithm as follows:

- **Input**: Description of a convex polygonal robot and a single convex polygonal obstacle, as a list of their vertices described in counter-clockwise order.
- **Output:** For 32 regularly spaced θ layers (starting at θ = 0 until θ = 2π−2π/32), compute the vertices of the c-obstacle slices, and draw the lines connecting these vertices.
- **Test:** Test your code on the robot A and the obstacle B_1 given in Figure 1. Give us a copy of your code and a printout of layers 1, 8, 16, 32.

Solution:

In order to solve this problem, we followed the algorithm for obstacles in configuration-space we learnt during lectures. We will now show step by step how we implemented it.

1. First of all, we defined the robot A and obstacle B_1 as a list of the vertices in the xy-plane (the robot position depends on a parameter θ that defines the rotation).

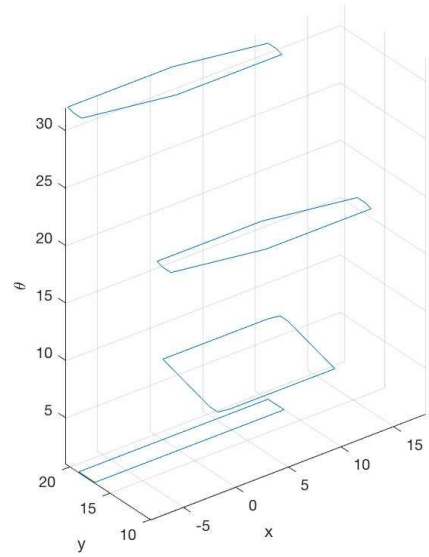
2. We calculated the normal vectors to each edge of the robot (normal going inwards) and of the obstacle (normal going outwards). Additionally, for each normal we add the value of its angle in the range $[0,2\pi]$. For more detail see the MATLAB function `normals(A)`. Note that the function only finds the normals pointing inside, therefore we had to change the direction for the obstacle's normals.

3. We place each normal across the unit-circle, ordered according to the angle.

4. For each sector in the unit-circle, defined by the normals of the robot and of the obstacle, we saved the corresponding value $b_i - a_j$ that defines the coordinate of the vertex of the $\mathcal{CB}$-space. Note that in general we expect to obtain $n_A + n_B$ vertices, i.e. in this case 8. However, for specific orientations (as for $\theta = 90 + \frac{\pi}{2}k$) we obtained less vertices, therefore we filled the empty entries with zeros, in order to obtain a matrix containing the full $\mathcal{CB}$-space.

For details regarding step 3 and 4, see function `IndCObstacle(A,B)`.

5. Run the aforementioned functions for 32 θ in the range $\left[0,2\pi - \frac{2\pi}{32}\right]$ and plot the $\mathcal{CB}$-space for each of them in the $xy\theta$-space. Here we show the results for the layers of θ number 1,8,16,32. (See matlab code for Question 1).

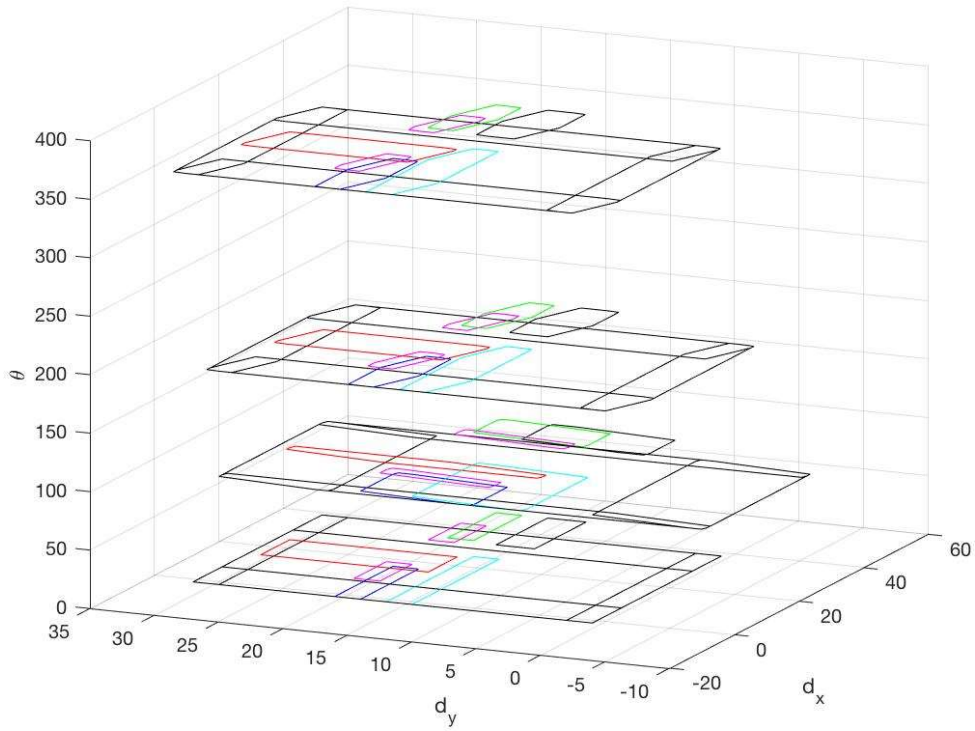


Part 2

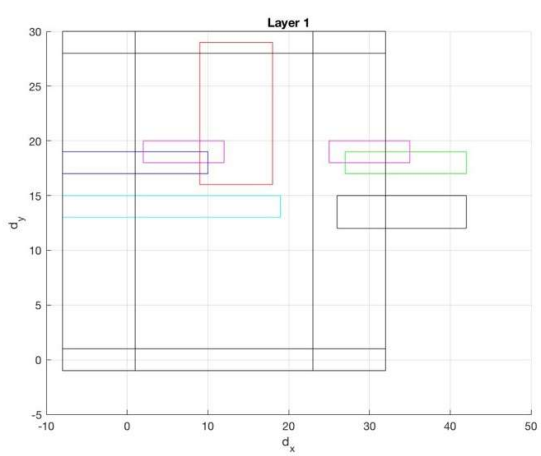
Extend your code to accommodate a list of possibly overlapping convex polygonal obstacles. Test your code on the example of Figure 1. The example shows an apartment with two bedrooms and a living room connected by a short corridor. The outer wall is represented by a union of four rectangular obstacles B_{0_1}, B_{0_2}, B_{0_3}, B_{0_4}. The interior walls are rectangular obstacles $B_1, \ldots, B_5$. The two doors are B_6, B_7.
Give us a printout of layers 1, 8, 16, 32. It would be nice if you mark in each slice the regions belonging to the respective physical obstacles.

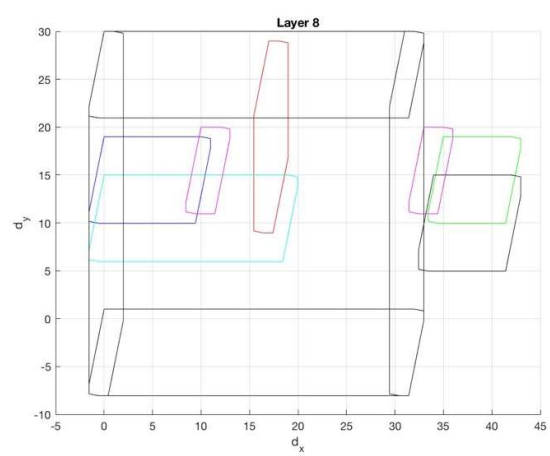
Solution:

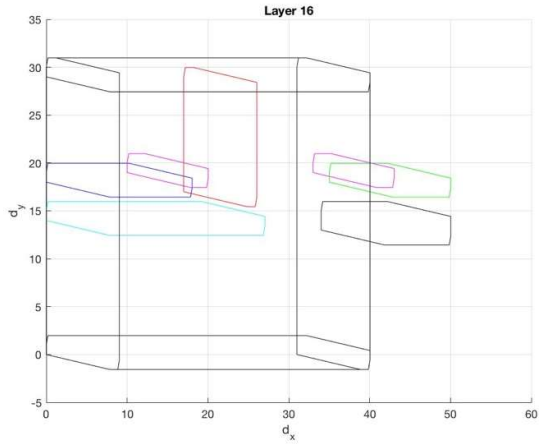
We save all the objects as a list of the vertices coordinates into a 3-dimensional matrix (`BuildObstale(A,B)`), Afterward, we run the code from Part 1 for each object. Finally we plot the $\mathcal{CB}$-space of all the obstacles (here it is showed only the layers of θ number 1,8,16,32. (See matlab code for Question 2).

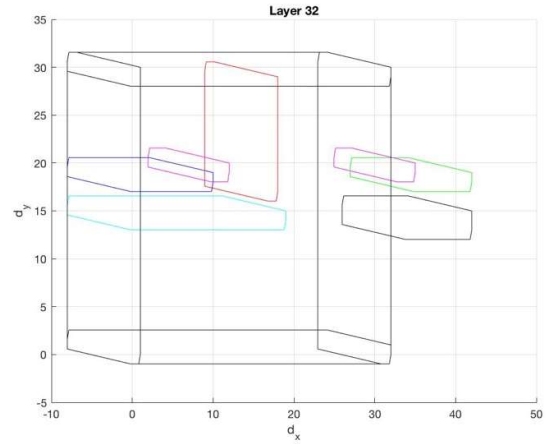


We can plot each layer separately:









Part 3

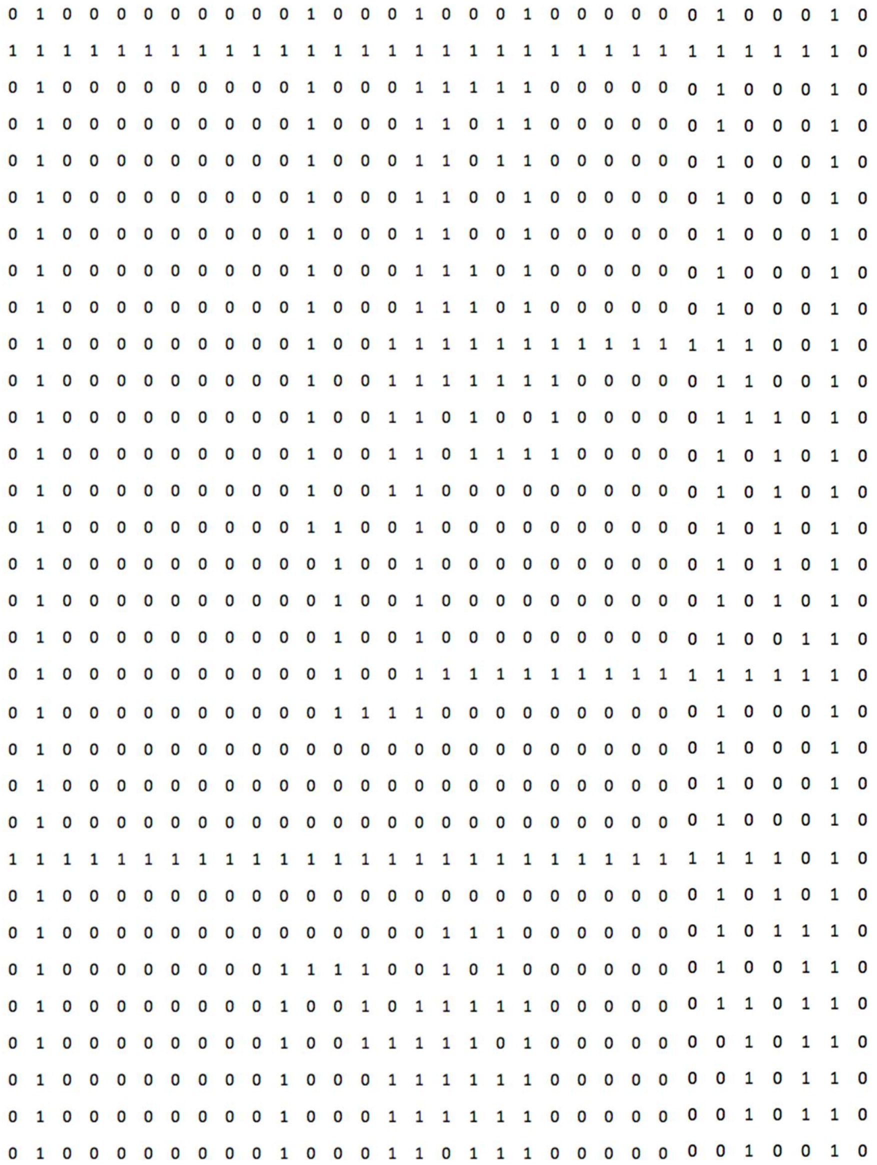
As a preparation for part #2 of the mini-project, set up an (x, y, θ) grid of dimension 32 × 32 × 32. For every θ layer, fill the cells of the respective c-obstacles boundaries by 1. The others should be filled with 0. Note that you will need a code that draws a discretized straight line between two given endpoints. Note, too, that portions of the c-obstacles corresponding to the outer wall spill out of the grid. Give us a printout of layers 1, 8, 16, 32.

Solution:

1. Firstly, we defined a $32 \times 32 \times 32$ matrix filled with zeros. every cell of the matrix represents a 1×1 square on the apartments floor.
2. For every rotation angle of the robot and for every obstacle, we took the points of CB_i that were found on the previous question and calculated a discrete set of points on the edge of the obstacle. We chose to use a small step between each point to get a good enough representation of all the squares the edge is occupying in the robots C-space.
3. For every point found on the edge of CB_i, we found the corresponding square it occupies.
4. Every occupied square we changed the square value from 0 to the number of the obstacle that occupies it. This way, we can see what obstacle occupies which square.

See the MATLAB code for Question 3

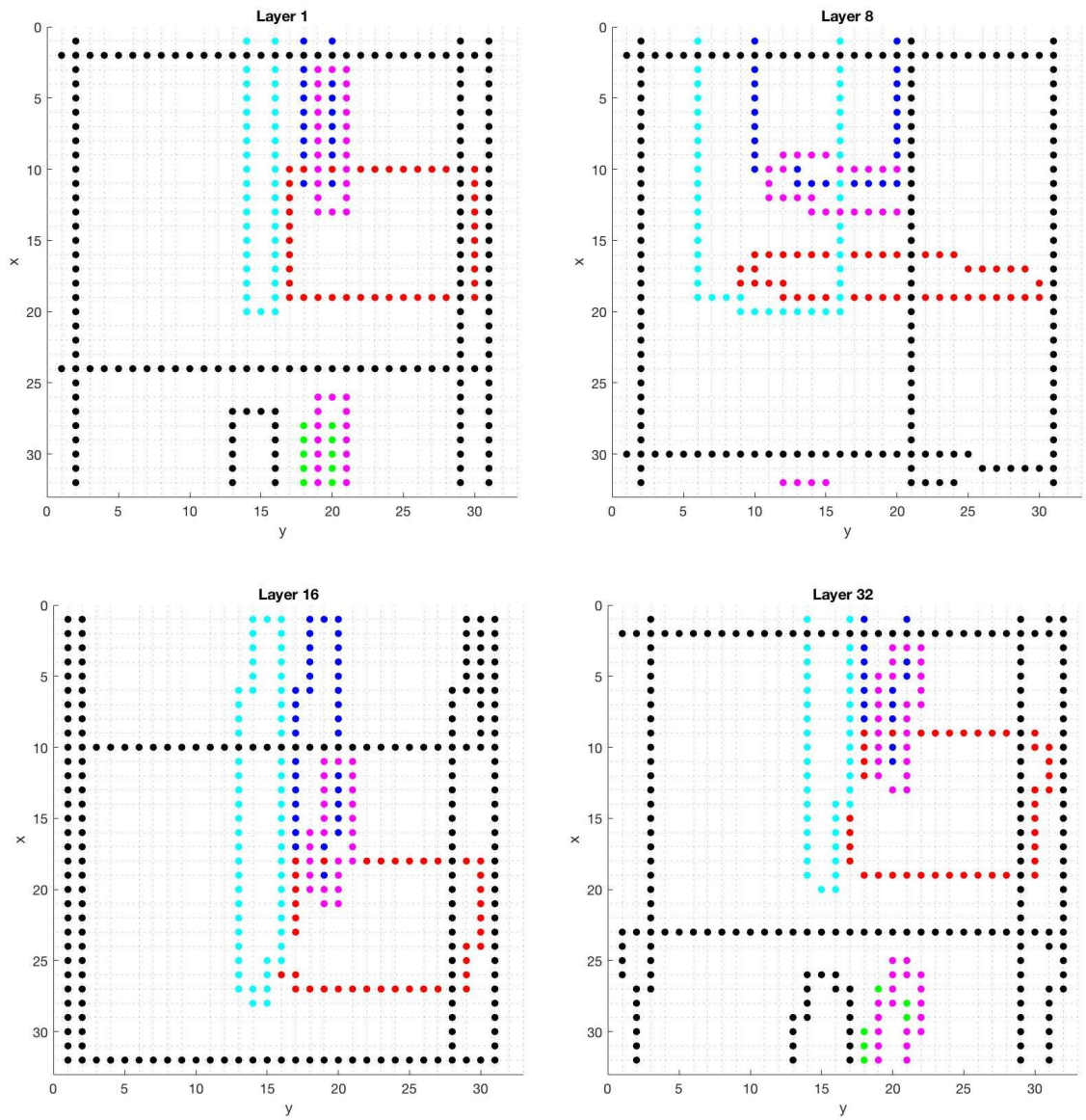
If we write on each cell that corresponds to an obstacle's boundary a 1, for example for the second layer, we obtain the following 32x32 matrix:



```
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0
0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0
0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 1 0
0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0 1 1 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0
0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 1 0
```

In order to clearly recognize each obstacle, we plot a dot in each cell occupied by a 1, while we leave empty the cells corresponding to a zero. Moreover, each color corresponds to a different object, so it is easy to see to which obstacle each boundary corresponds.

We plot each layer separately:



Note that there might be boundaries from different obstacles corresponding to the same point, in this case the point is marked in one of the colors only.